Discrete Applied Mathematics 159 (2011) 1118–1130

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Facility location problems: A parameterized view

Michael R. Fellows [a], Henning Fernau [b,*]

[a] Charles Darwin University, Darwin, Northern Territory 0909, Australia

[b] Universität Trier, FB IV—Abteilung Informatik, 54286 Trier, Germany

ARTICLE INFO

Article history:
Received 15 September 2008
Received in revised form 10 March 2011
Accepted 25 March 2011
Available online 21 April 2011

Keywords:
Exact algorithms
Parameterized algorithms
$\mathcal{NP}$-completeness
Facility location problems
Semistructured documents

ABSTRACT

Facility location problems have been investigated in the Operations Research literature from a variety of algorithmic perspectives, including those of approximation algorithms, heuristics, and linear programming. We introduce the study of these problems from the point of view of parameterized algorithms and complexity. Some applications of algorithms for these problems in the processing of semistructured documents and in computational biology are also described.

1. Introduction

Operations Research offers an extensive literature concerned with several formalizations and variants of *facility location problems*. Such problems model the following scenario:

> A company wants to open up a number of facilities to serve their customers. Both the opening of a facility at a specific location, and the service of a particular customer through a particular facility, incurs some cost. The goal is to minimize the overall cost of opening enough facilities to serve all the customers.

We introduce the investigation of these problems from the perspective of parameterized complexity and algorithms. Properly formalized, facility location problems can also be used to model algorithmic issues in other application areas of Operations Research, Computer Science and Computational Biology.

Most variants of the class of problems that we are concerned with are $\mathcal{NP}$-hard. Motivated by their significant applications, attempts have been made to devise useful algorithms, according to three well-known strategies for "coping with $\mathcal{NP}$-hardness":

(1) Heuristics that are guaranteed to run in reasonable time and produce a solution—but with no mathematical guarantee concerning its quality.
(2) Approximation algorithms that run in reasonable time and produce a solution guaranteed to be within some distance of optimal (usually a very wide distance).
(3) A reformulation of the problem as an integer (non-)linear programming task. LP-solvers are then applied. This approach differs from (1) in that, in a reasonable amount of time, one might not get any answer, but if a solution is produced, it is optimal. This third approach has proved to be relatively successful, in part because LP-solvers have been extensively studied both in theory and in practice.

* Corresponding author.
E-mail address: fernau@uni-trier.de (H. Fernau).

0166-218X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2011.03.021

Our research in this paper is aligned with (3). We offer an exploration of the parameterized complexity of this class of problems. We describe an abstract form of the problem, and investigate its complexity under several natural parameterizations. We also exposit how this general form of the problem connects to other applications that are seemingly far removed from locating physical facilities such as stores.

Parameterized complexity can be viewed as a multivariate generalization of the familiar *P versus NP* framework that is essentially *one-dimensional*. In this familiar framework, complexity is assessed, in the worst case, over all inputs of size (the one measurement) n. This has proven to be far too pessimistic for real-world input distributions for many computational problems. Natural input distributions tend to have important *secondary structure* that significantly affects the practical computational complexity of the problem. In the framework of parameterized complexity, the *parameter* captures the relevant secondary structure. Background on parameterized complexity can be found below.

1.1. Definitions

We study the following problem of Facility Location and variants thereof:

Given: A bipartite graph $B = (F \uplus C, E)$, consisting of a set F of potential facility locations, a set C of customers, and an edge relation E, where $\{f, c\} \in E$ indicates that c can be served from the facility (at) f; weight functions $\omega_F : F \to \mathbb{N}_{\geq 1}$ and $\omega_E : E \to \mathbb{N}_{\geq 1}$ (both called ω if no confusion may arise) that model the costs of building facilities at various locations, and the costs of serving the customers from facilities at those locations $k \in \mathbb{N}$ representing the total budget.

Question: Is there a set $F' \subseteq F$ of facility locations and a set $E' \subseteq E$ of ways to serve customers such that:

(1) every edge in E' is incident on a vertex in F', a requirement that expresses that edges used to serve customers must come from locations chosen for the facilities, or formally, $\forall f \in F(f \in F' \iff \exists e \in E'(f \in e))$,
(2) every customer is served by some edge in E', or formally, $\forall c \in C \exists e \in E'(c \in e)$, and
(3) that the budget is observed, expressed formally, $\sum_{f \in F'} \omega_F(f) + \sum_{e \in E'} \omega_E(e) \leq k$?

The set F' represents in a solution the locations where facilities are to be opened. The set E' represents how the customers should be served from those facilities.

In the literature, the problem formulated above is mostly known as the uncapacitated discrete facility location problem. See [5] for a recent overview.

Alternatively, and sometimes more conveniently, this problem can also be formulated in terms of a matrix representation of the facility location and customer service costs.

Facility Location (matrix formulation)

Given: A matrix $M \in \mathbb{N}_{\geq 1}^{(n+1)\times m}$, indexed as $M[0 \ldots n][1 \ldots m]$ $k \in \mathbb{N}$.

Question: Is there a set $F' \subseteq \{1, \ldots, m\}$ of columns and a service function $s : \{1, \ldots, n\} \to F'$ such that $\left(\sum_{f \in F'} M[0, f]\right) + \left(\sum_{c=1}^{n} M[c, s(c)]\right) \leq k$?

In the matrix formulation, the columns play the role of the m potential facility locations and the rows represent the n customers to be served (except for row 0).

Edges that are not present in the bipartite graph formulation can be modeled in the matrix representation as edges that have a weight larger than k. The matrix $M[1 \ldots n][1 \ldots m]$ records the weights of the edges, while $M[0][1 \ldots m]$ records the weights associated with potential facility locations. Condition (2) in the bipartite graph formulation can be used to construct the service function s. The equivalence of the formulations is straightforward. In the following, we will use terminology from the two formulations interchangeably, according to convenience.

1.2. Fixed parameter tractability

$\mathcal{NP}$-hard computational problems are ubiquitous in economics. One approach to overcoming this difficulty is to devise algorithms that can solve arbitrary instances of such a problem under the restriction that a relevant secondary measurement, called the *parameter*, is small.

This concept is formalized as follows. Problem instances are elements of $\Sigma^* \times \mathbb{N}$, and an instance $I = (w, k)$ is to be decided in time $\mathcal{O}(p(|w|)f(k))$, where p is a polynomial (whose degree does not depend on the parameter k) and f is an arbitrary function. Problems that can be solved within such a time restriction are said to be *fixed parameter tractable*, or in $\mathcal{FPT}$. Equivalently, a problem is in $\mathcal{FPT}$ iff there exists a polynomial time computable self-reduction κ that maps an instance $I = (w, k)$ onto an(other) instance $I' = (w', k')$ of the same problem whose overall size is limited by a function $g(k)$, i.e., $|w'| + k' \leq g(k)$. Then, I' is also called a *problem kernel* for I, and κ is the corresponding *kernelization*. There is also a complementary intractability theory, reflected in the W-hierarchy of parameterized problem classes

$$\mathcal{FPT} = W[0] \subseteq W[1] \subseteq W[2] \subseteq \ldots,$$

and an appropriate notion of parameterized problem reduction. $W[1]$-hardness corresponds to $\mathcal{NP}$-hardness in classical complexity theory. Further details can be found in the textbook [14].

The $\mathcal{O}^*$-notation extends the familiar $\mathcal{O}$-notation by suppressing factors that are polynomial in the input size n. This gives a convenient shorthand for stating results in exact exponential time and parameterized algorithmics. In particular, a

parameterized problem (with parameter k) is in $\mathcal{FPT}$ if and only if it can be solved in time $\mathcal{O}^*(f(k))$, where f is an arbitrary function.

1.3. Our contribution

Facility location problems have been extensively studied in the literature from a variety of algorithmic perspectives, including hardness, approximation, and heuristics. Our aim is to initiate systematic research on these problems from the viewpoint of parameterized complexity. We will show parameterized tractability for the problem formulation described and many variants thereof, and also prove some parameterized intractability results. We provide an introduction to how $\mathcal{FPT}$-results can be obtained in a systematic fashion, improving on initial $\mathcal{FPT}$-classification results obtained by various techniques.

As a secondary objective, we exhibit connections to several application areas outside economics, which might stir up research uniting different communities on this topic. In particular, we will focus on a learning problem that arises when dealing with semistructured documents.

1.4. Related work and variations

It is often useful to relate the costs of serving customers to an underlying metric space, i.e., $\omega_E(c, f)$ can be described in terms of the distance of c and f times the demand that is incurred by c. This problem variant is also referred to as the (metric) uncapacitated facility location problem. A more general setting is given in the capacitated facility location problem, where each facility can only serve up to a given maximum load; then, there are again two variants according to whether it is allowed that the demand of a customer may be satisfied from only one facility, or whether it is to be split among different facilities. Good overviews of approximation algorithms for many variants can be found in [5,25,27].

2. Facility Location is in $\mathcal{FPT}$

We describe several approaches to classifying the problem in $\mathcal{FPT}$. This can be taken as a short introduction to the whole field of parameterized algorithmics, following this specific example.

One of the first things that has to be decided is what parameters (as a secondary measurement of the whole input) are to be considered. In our case, natural choices could be: (1) the number n of customers, (2) the number m of potential facility locations, (3) an upper bound k on the cost, or (4) an upper bound ℓ on the number of facilities that could be opened.

A trivial brute-force approach immediately yields:

Theorem 1. Facility Location *can be solved in time* $\mathcal{O}^*(2^m)$. □

Parameter m is not particularly interesting, as long as we are mainly interested in classification results. For the other parameters suggested, the situation is less clear. We put special emphasis on k, since this is the choice of parameter that is usually considered natural for minimization problems.

2.1. Finding reduction rules

Kernelization models preprocessing. A kernelization algorithm is a polynomial time transformation that maps a problem instance (I, k) to (I', k') such that the size of the new instance (I', k') is bounded by a function of k. A parameterized problem is in $\mathcal{FPT}$ if and only if it admits a kernelization algorithm. Kernelization algorithms are typically based on locally defined reduction rules. The quest for reduction rules is essential in parameterized algorithmics. We begin with the following easy observation.

Reduction Rule 1. If a given instance (M, k) with $M \in \mathbb{N}_{\geq 1}^{(n+1)\times m}$ obeys $n > k$, then return NO.

Lemma 1. *Rule 1 is valid.*

Proof. Each customer must be served. Since edges have positive integer weights, each customer thus incurs "serving costs" of at least one unit. Hence, no more than k customers can be served. □

Lemma 2. *After exhaustive application of Rule 1, the reduced instance* (M, k) *has no more than* $(k + 1)$ *rows.* □

Notice that the preceding lemma builds a close link between the parameters k and n.

A facility location f is described by the vector $v_f = M[0 \ldots n][f]$. These vectors can be compared componentwise; let $\leq$ denote the according partial order.

Reduction Rule 2. If for two facility locations f and g, $v_f \leq v_g$, then delete g; the parameter stays the same.

Notice that the expression "delete g" means removing g from the bipartite graph model; in the matrix model, this corresponds to deleting the row indexed g.

Lemma 3. *Rule 2 is sound.*

Proof. Obviously, a solution to the Rule-2-reduced instance is also a solution to the original instance. If we had a solution S to the original instance that contains a facility g and there is another facility f such that $v_f \leq v_g$, then a solution S' obtained from S by choosing facility f instead of g (and choosing to serve any customer served by f in S to be served by g in S') will also be a valid solution of no greater cost. □

Reduction Rule 3. Consider an instance $((B, \omega_E, \omega_F), k)$ of Facility Location. The following modifications will not affect the parameter.
Forall facilities f **do**
(1) If $\omega_F(f) \geq k$, then delete f.
Forall customers c **do**
(2) If $\omega_F(f) + \omega_E(c, f) > k + 1$, then set $\omega_E(c, f) := k + 1 - \omega_F(f)$.

Lemma 4. *Rule 3 is sound.*

Proof. A facility with opening costs of at least k cannot serve anybody at all (since serving only one customer in this way will incur total costs of at least $k+1$). Therefore, such expensive facility locations should not be considered at all. If a certain facility is not too expensive by itself, it might still incur exorbitant serving costs. However, in that case, we only need to store the impossibility of such serving, which is already reflected by having $\omega_E(c, f) + \omega_F(f) = k + 1$. This explains the second part of the rule. □

2.2. Kernelization through well-quasi-orderings

Central to the complexity class $\mathcal{FPT}$ is the concept of kernelization, since it characterizes $\mathcal{FPT}$. We first provide a "quick classification" of Facility Location in $\mathcal{FPT}$, based on well-quasi-orderings.

Theorem 2. Facility Location *is fixed parameter tractable when parameterized by an upper bound k on the cost.*

Proof. Let (M, k) be reduced with respect to Rules 1–3. By Lemma 2, we know that M contains no more than $(k + 1)$ rows, since (M, k) is Rule-1-reduced. Each facility is therefore characterized by a $(k+1)$-dimensional vector. Since (M, k) is Rule-2-reduced, all these vectors are pairwise incomparable. By Dickson's lemma [11], the number of vectors is bounded by some function $g(k)$, since (M, k) is Rule-3-reduced. Hence, M is a matrix with no more than $(k + 1)g(k)$ entries. □

The function $f(k) = (k + 1)g(k)$ derived for the kernel size in the previous theorem is huge, yet it provides the required classification. More precisely, consider $g(k)$, upper bounding the number of pairwise uncomparable $(k + 1)$-dimensional vectors of natural numbers, each component being upper bounded by $k + 1$ according to Rule 3. Clearly, there are no more than $(k + 2)^{k+1}$ many vectors of natural numbers in the range between 0 and $k + 1$ at all, but the geometric argument illuminating Dickson's lemma in [11, Pages 414–415] and in [24, Pages 37–28] for the two-dimensional case indicates that this is in fact already the worst case, asymptotically speaking. However, in practical terms, we might expect this quantity to be significantly smaller. These considerations are continued in Lemma 6.

We can use Theorem 1 and the reasoning in the previous paragraph to show:

Corollary 1. Facility Location *can be solved in time* $\mathcal{O}^*(2^{(k+2)^{k+1}})$. □

In the following, we improve on this (quick) $\mathcal{FPT}$-classification result.

2.3. Kernelization refinements

To obtain a better algorithm, observe that a solution can be viewed as a partition of the set of all customers into groups such that customers within the same group are served by the same facility. A solution specified by the selected facilities and selected serving connections can be readily transformed into this sort of partition. Also the converse is true: given a partition of the set of customers, we can compute in polynomial time the cheapest way to serve this group by means of a certain facility.

This observation immediately yields:

Lemma 5. *On an instance* (M, k), *where* $M \in \mathbb{N}_{\geq 1}^{(n+1)\times m}$, Facility Location *can be solved in time* $\mathcal{O}(k^k p(g(k)) + nm)$, *where* p *is some polynomial and* $g(k)$ *bounds the number of facilities.*

Proof. The kernelization rules 1 and 2 can be applied in time $\mathcal{O}(nm)$. Then, we have to check all partitions; there are $o(k^k)$ many of them. (More precisely, the number is described by Bell's number, whose asymptotics is due to de Brujn (1958)[1].) For each partition, we compute its cost based on the assumption that each class of the partition is served by one facility; this can be done in polynomial time. □

Lemma 6. *An instance* $(M, k), M \in \mathbb{N}_{\geq 1}^{(n+1)\times m}$, *of* Facility Location *obeys* $nm \leq (k + 1)^{k+2}$ *if it is reduced with respect to Rules 1–3.*

[1] For further information, see http://mathworld.wolfram.com/BellNumber.html.

Proof. Let (M, k) be such a reduced instance. Due to Rule 1, there are at most k customers. Due to Rule 3, there are at most $(k+1)^{k+1}$ different facility vectors. Due to Rule 2, vectors of different facilities must be different. Hence, there are at most $(k+1)^{k+1}$ facilities. Since each facility vector has at most $k+1$ entries, M has no more than $(k+1)^{k+2}$ many entries. □

One might argue that the kernel size obtained is quite huge and not very useful. So, there are two natural questions:

- Are there any kernels for this problem that are much smaller than what we obtained?
- Are there any parameterized algorithms different from running a brute-force algorithm on the kernel?

2.4. Lower bounds on kernel sizes

To answer the first question, a certain machinery has been developed in recent times that we can make use of. We need a few notions and results from the theory of kernel lower bounds [3,12,18]. We start off by defining another problem:

Red–Blue Dominating Set (RBDS).

Given: An undirected bipartite graph $G = (R \uplus B, E)$, and a positive integer k.

Parameter: $k + |B|$

Question: Does there exist a set $D \subseteq R$ of at most k vertices of G such that every $v \in B$ is adjacent to some $u \in D$ (i.e., D is a dominating set of B)?

Dom et al. have shown [12] that RBDS, parameterized by $(k + |B|)$, does not admit a polynomial kernel unless the Polynomial Time Hierarchy collapses to the third level. We will make use of that result, on the basis of the following notions; see [4]:

- Let $\Pi \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let $1 \notin \Sigma$ be a fresh symbol. We define the *derived classical problem* associated with Π to be $\left\{x1^k \mid (x, k) \in \Pi\right\}$.
- Let P and Q be parameterized problems. We say that P is *polynomial time and parameter reducible* to Q if there exists a polynomial time computable function $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$, and a polynomial $p : \mathbb{N} \to \mathbb{N}$, and for all $x \in \Sigma^*$ and $k \in \mathbb{N}$, if $f((x, k)) = \left(x', k'\right)$, then $(x, k) \in P$ if and only if $\left(x', k'\right) \in Q$, and $k' \leq p(k)$. We call f a *polynomial time and parameter transformation* (or a *PTP reduction*) from P to Q.

This notion of a reduction is useful in showing kernel lower bounds because of [4, Theorem 3] that can be phrased as follows. Let P and Q be parameterized problems whose derived classical problems are P^c, Q^c, respectively. Let P^c be $\mathcal{NP}$-complete, and $Q^c \in \mathcal{NP}$. Suppose there exists a PTP reduction from P to Q. Then, if Q has a polynomial kernel, then P has a polynomial kernel.

Theorem 3. Facility Location *has no kernel of size bounded by k^c, for any fixed constant c, unless the Polynomial Time Hierarchy collapses to the third level.*

Proof. Consider an undirected bipartite graph $G = (R \uplus B, E)$. If one associates the red (R) vertices with (potential) facility locations and the blue (B) vertices with customers, associating cost 1 to edges in E and cost $k + |B| + 1$ to vertex pairs $\{r, b\} \notin E$ with $r \in R$ and $b \in B$, as well as unit cost to opening up any facility, then there exists a set $D \subseteq R$ of at most k vertices of G such that every $v \in B$ is adjacent to some $u \in D$ if and only if there is a set of facilities $F \subseteq R$ serving all customers through a set of service edges $E' \subseteq E$ with $|E'| + |F| \leq k + |B|$.

$\rightarrow$: Let $F = D$ and E' be any selection from E that shows that all B-vertices are dominated (or "served").

$\leftarrow$: Clearly, edges not in E are too expensive to be service edges. Hence, $E' \subseteq E$. Since each B-vertex (customer) is served, $|B| \leq |E'|$. Hence, F dominates B with $|F| \leq k$, so we can set $D = F$.

We have presented a PTP reduction from RBDS to Facility Location. Hence, a polynomial kernel for Facility Location would translate into a polynomial kernel for RBDS, contradicting [12, Theorem 2]. □

2.5. Improving on brute force by using dynamic programming

The idea of "dynamic programming on subsets" improves the running time.

Theorem 4. Facility Location *can be solved in time $\mathcal{O}(2^k m + 3^k)$ on a given instance (M, k) with $M \in \mathbb{N}_{\geq 1}^{(n+1)\times m}$.*

Proof. Recall that in the matrix formulation, the columns play the role of the m potential facility locations and the rows represent the n customers to be served (except for row 0). We start with Rule 1. So, we know that $n \leq k$. In a preprocessing phase, we compute the lowest cost incurred by a certain set X of customers when being served by a single facility, storing the results in a table "one-serve" (**OS**) with $2^n \leq 2^k$ entries. This also includes the costs for opening up that facility. More precisely, first compute for X the auxiliary table F_X such that $F_X(f)$ stores the cost of serving all of X through $f \in F$. Then, store in **OS**(X) the minimum of all entries in $F_X(f)$.

Algorithm 1 A dynamic programming algorithm for facility location: FLdp

Require: a bipartite graph $B = (F \uplus C, E)$ with weights ω_E and ω_F
Ensure: an implicit facility location strategy incurring minimum weight

```
if |C| > k then
  return NO;
s(∅) := 0;
for all X ⊆ C do
  compute OS(X);{in time O(|X| · |F|)}
for i := 1, ..., |C| do
  for all X ⊆ C, |X| = i do
    s(X) := min_{∅⊊Y⊆X}(OS(Y) + s(X \ Y))
    {Every customer belongs to Y, to X \ Y or to C \ X;}
    {Convention: min_∅ ... = ∞.}
```

Then, we can compute the minimal costs of serving a certain group of customers by using some facilities by dynamic programming, combining two subsets at a time. If we have stored the results of the preprocessing in table **OS**, each step in the dynamic programming will take only constant time, so we arrive at the following formula for dynamic programming:

$$s(X) := \min_{\emptyset \subsetneq Y \subseteq X} (\mathbf{OS}(Y) + s(X \setminus Y)). \tag{1}$$

This amounts to $\mathcal{O}(3^n) \subseteq \mathcal{O}(3^k)$ operations. Namely, there are basically three possibilities for a customer: it belongs to Y, $X \setminus Y$ or $C \setminus X$. This (and the correctness of the procedure) can be seen as follows.

Notice that when $|X| = 1$, then $Y = X$ is enforced in the formula (1) for computing $s(X)$, so $s(X) = \mathbf{OS}(X) + s(\emptyset) = \mathbf{OS}(X)$ is computed. In later stages of the recursion, it should be immediate that the formula is correct if the facility f used to serve Y is not among the facilities used to serve $X \setminus Y$ (*). However, if this is not the case, we certainly did not find the minimum: a smaller value avoiding the double counting of the opening cost of f is obtained as follows. Let Y' denote those customers in $X \setminus Y$ that are also served by f. Then,

$$s(X) \leq \mathbf{OS}(Y \cup Y') + s(X \setminus (Y \cup Y')) \leq \mathbf{OS}(Y) + s(X \setminus Y).$$

Hence, it is sufficient to consider condition (*). □

Remark 1. Notice that the proof of the preceding theorem immediately implies an $\mathcal{O}^*(3^n)$ estimate for Algorithm 1, since m is "swallowed" as the polynomial part of the overall input size. Alternatively, we can first kernelize (with respect to k) and then run Algorithm 1 on the kernel, which would amount to an $\mathcal{O}^*((2k)^k)$ estimate due to Lemma 6.

2.6. Further improvements

In a very recent paper, Björklund et al. [1] described how to further speed up dynamic programming on subsets in just such a situation as is encountered with our problem. According to their approach the recursion from Eq. (1) can be seen as a subset convolution within the min-sum semiring of integers. This is then interpreted as a subset convolution over the sum–product ring, which can be solved fast via the fast zeta and Möbius transforms. Let us briefly explain this approach (with n customers and m facilities). By computing first the ranked Möbius transform (which basically takes 2^n steps), this convolution operation can be performed in the transformed space with $2n+1$ operations, and the inverse Möbius transform is of similar cost to the Möbius transform itself.

Corollary 2. *If all the integer weights lie between 1 and N for a given instance of* Minimum Facility Location*, the problem can be solved in time* $\mathcal{O}(2^n n^3 (nm)^2 \log(N))$. □

The assumption of a bounded range of integer weights is not so unrealistic in many scenarios; for example, in our parameterized setting, we can assume (by our reduction rules) that those weights lie between 1 and $k + 1$.

3. Variants of Facility Location

In this section, we discuss several variants of Facility Location that are frequently discussed in the literature. These variations sometimes generalize the framework that we presented, and sometimes they can be seen as special cases.

3.1. The median/means problem

The p-median problem and the p-means problem are defined just like the Facility Location problem, except for the fact that there are no costs for opening up facilities (and mostly, it is required that exactly p facilities should open). Means and median problems only differ in the metric, i.e., the way point distances are measured (sums of squares of distances versus sums of distances). All corresponding problems are $\mathcal{NP}$-hard, even for Euclidean spaces. Obviously, these problems have

quite a geometric flavor. Again, it is not hard to see by a simple analysis of our results of the preceding section that also the p-MEDIAN PROBLEM and the p-MEANS PROBLEM are in $\mathcal{FPT}$: namely, even the common generalization of these problems, viewed as variants of FACILITY LOCATION, where the opening cost of a facility might be zero, is in $\mathcal{FPT}$.

Many approximation algorithms are known here; see for example [8] for the p-MEDIAN PROBLEM. Often in this literature the concern is with a variant of the uncapacitated metric facility location problem. Here, the costs for connecting facilities and customers are distances in a metric space. If the possible facility locations are not distinguished from customer locations, then this models the problem of finding a minimum cost clustering in a metric space. Our preceding arguments also show $\mathcal{FPT}$-membership in this case.

3.2. Rational weights

In all versions discussed up to now, including the one introduced at the very beginning, one could also allow rational (or even "real-number") weights bigger than or equal to 1 (modeling actual costs more realistically). Apart from numerical considerations, this would not change anything except for the sections dealing with Dickson's lemma and the one dealing with the fast subset convolution; those approaches would not work here. On the positive side, the dynamic programming approach detailed in the preceding section can be easily adapted.

3.3. Cheap serves

What happens if we allow services of zero cost? Irrespectively of whether or not zero costs for opening up facilities are allowed, the complexity picture changes dramatically.

We describe parameterized reductions to and from the following problems[2]:

HITTING SET (HS).

Given: A hypergraph $G = (V, E)$.

Parameter: A positive integer k.

Question: Is there a *hitting set* $C \subseteq V$ with $|C| \leq k$?

SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION (SMNTMC).

Given: A multi-tape nondeterministic Turing machine M (with two-way infinite tapes), and an input string x.

Parameter: A positive integer k.

Question: Is there an accepting computation of M on input x that reaches a final accepting state in at most k steps?

Theorem 5 (*See [14]*)**.** HITTING SET *and* SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION *are* $W[2]$*-complete.*

Theorem 6. FACILITY LOCATION*, parameterized by* k*, becomes* $W[2]$*-complete when zero-weight services are allowed.*

Proof. We first show that FACILITY LOCATION, parameterized by k, becomes $W[2]$-hard when zero-weight services are allowed. We use HS in this reduction. Consider a hypergraph $G = (V, E)$. Call the elements of V facilities and the elements of E customers. More precisely, a facility v may serve a customer e if and only if $v \in e$. Assume that opening any facility costs one unit, and all customer services are for free, i.e., they incur zero cost. Then, the hitting set problem corresponds to selecting (at most) k of the facilities to serve all customers.

Conversely, membership in $W[2]$ can be seen, e.g., by showing how to solve our problem on an instance $B = (F \uplus C, E)$ with weight function ω by a multi-tape nondeterministic Turing machine with empty input; see [7]. We only sketch the work of such a machine M in the following. M contains $|C| + 2$ tapes, forming the tape set $T = \{g, b\} \cup C$. At the very beginning, M writes a left-end marker on all tapes. In a first phase, M guesses at most k facility locations and writes these guesses on the guessing tape g, using a specific character per location. In a second phase, M reads g and moves the head on the budget tape b by $\omega(f)$ steps to the right when scanning (the character of) facility f on g. In a third phase, M reads through g again and does the following (in parallel) for each C-tape c: if the current symbol f on g can serve customer c at a cost of ω, then the head on c moves to the right and writes a special symbol for ω, provided that the head on c was previously reading a left-end marker; otherwise, it would compare the value ω' previously needed to serve c with ω and store the lesser cost on c. In a fourth phase, it is tested (in parallel) if any C-tape still scans the left-end marker, which would mean that we have guessed a set of facility locations that cannot serve all customers at all. If such an inconsistency is detected, M enters an endless loop. In a fifth phase, at most k customers are guessed (using the guessing tape g again) that are served using non-zero costs. In a sixth phase, for each of the guessed customers c, we scan the value $\omega(c)$ that has been computed; the head on the budget tape b is moved $\omega(c)$ steps to the right, and the symbol on tape c is rewritten as zero. Finally, we verify that the head on the budget tape has moved at most k steps to the right in total, and that all C-tape heads scan zero. Only if these final tests succeed does M halt. Hence, M halts in $f(k)$ steps for some function f linear in k if and only if the given FACILITY LOCATION instance has a solution. □

[2] See [14] for details on parameterized reductions.

Recall that, according to the preceding section, ℓ denotes an upper bound on the number of facilities that could be opened, producing another natural parameter of the problem. The proof of the preceding theorem entails:

Theorem 7. FACILITY LOCATION, *parameterized by* ℓ, *is* $W[2]$*-complete.*

4. Some surprising applications: locating facilities and the MDL principle

FACILITY LOCATION and its variants have numerous applications, even when only allowing integer costs. We describe some of the less obvious ones based on the Minimum Description Length (MDL) principle.

4.1. Automating the production of XML documents

The web standard exchange format XML is described in: Extensible Markup Language (XML) 1.0, Bray, Paoli, and Sperberg-McQueen, 10 February 1998, available at http://www.w3.org/TR/REC-xml. In his notes on this standard, Bray wrote,[3] commenting on the construction of document type descriptors (DTD), a context-free grammar used to specify syntactic characteristics of XML documents:

> "Suppose you're *given an existing well-formed XML document* and you want to build a DTD for it. One way to do this is as follows:
>
> 1. Make a list of all the element types that actually appear in the document, and *build a simple DTD* which declares each and every one of them as ANY. Now you've got a DTD (not a very useful one) and a valid document.
> 2. Pick one of the elements, and work out how it's actually used in the document. Design a rule, and *replace the ANY declaration with a …content declaration. This, of course, is the tricky part*, particularly in a large document.
> 3. Repeat step 2, working through the elements one by one, until you have a useful DTD."

Various systems, known as *DTD generators*, were designed to automate the process of designing DTDs, on the basis of examples. The main problem is that of *generalization*: when given a number of sample documents which should fit the envisaged DTD, at what "moment" and in what way should the DTD generator switch from a mode where it only produces a trivial DTD (this could be either a very specific one that can only parse the given samples, or a very general one, as attempted by Bray with his ANY declaration proposal) to a mode where it actually tries to cleverly guess the syntactical structure in the given documents? This process of making "intelligent guesses" is known as *generalization*.

Garofalakis et al. proposed for this purpose the system XTRACT [21], developed at Bell Labs and Stanford University (among other places). This DTD generator is based on the *Minimum Description Length (MDL) principle*, that can be viewed as a formalization of a line of reasoning commonly known as Occam's razor. This principle, in connection with grammar induction (DTD generators are a special case of this field), was discussed earlier by Conkley and Witten in [10]. The *length of a description* consists of two parts:

1. the length of the theory (in bits) and
2. the length of the data (in bits) when encoded with the help of the theory.

The system XTRACT uses MDL to evaluate regular expression hypotheses. This yields the combinatorial problem MDL-OPTIMAL-CODING:

Given: A set $R = \{r_1, \ldots, r_m\}$ of regular expressions (over the basic alphabet Σ) and a set of strings $S = \{s_1, \ldots, s_n\} \subset \Sigma^+$, $k \in \mathbb{N}$.

Question: Does there exist a subset R' of R such that

$$\sum_{r \in R'} |c(r)| + \sum_{s \in S} |c(s|R')| \leq k?$$

The coding function c is described in [21] and is further explained in the Appendix.

Theorem 8 (*Announced in [17]*)**.** MDL-OPTIMAL-CODING *is* $\mathcal{NP}$*-complete.*

Proof. [4] It is well-known that it is $\mathcal{NP}$-hard to compute, for any given cubic graph $G = (V, E)$, whether or not G has a vertex cover set of size at most k; see [19]. Consider such an instance $G = (V, E)$, together with k. We reduce to an equivalent MDL instance.

Let $\Sigma = E$ be the alphabet, serving also as our finite set of words $S(=E)$. With each vertex v, we associate the regular expression $r_v = e_0 \cup e_1 \cup e_2$, where e_0, e_1, e_2 are the three edges incident to v. This defines our set of regular expressions $R = \{r_v \mid v \in V\}$.

[3] Up to recently, this could be retrieved and read at the following webpage: www.xml.com/axml/notes.

[4] The proof presented here differs from the one presented at the poster session of ICGI 2004 [17].

Let R_C be the corresponding set of expressions. Assume that a vertex cover C of G of size k exists. Hence, the theory itself can be described by $5c_\Sigma k$ many bits. By construction, once r_v containing e_i is selected, e_i can be described by a constant number c of bits. Hence, the description of S needs $c\lceil\log_2(k)\rceil|E|$ additional bits. Setting $k' = 5c_\Sigma k + c\lceil\log_2(k)\rceil|E|$ completes the description.

The converse is also easily seen, i.e., if the MDL-OPTIMAL-CODING instance as described is given and has a solution of size k', then, by the special form of the instance, $k' = 5c_\Sigma k + c\lceil\log_2(k)\rceil|E|$ for some constant k, and this k corresponds to the number of regular expressions selected for coding. In order to be able to encode all $S = E$, the selected regular expressions correspond to a vertex cover of size k in the original graph. □

Garofalakis et al. [21] propose using the following translation to a related facility location problem: (a) place *facilities* (in our case: the selected theory) (b) into some *locations* (here: the regular expressions to choose from) (c) in order to minimize the *costs* incurred by the facilities (here: the number of bits needed to encode the theory) and by serving a given set of customers (in this case: the number of bits for encoding the given strings).

Our results imply that we can solve MDL-OPTIMAL-CODING as a parameterized problem to optimality. This is particularly important if MDL is actually used to *evaluate* hypotheses. Using here algorithms which only provide a logarithmic approximation guarantee will eventually mean that hypotheses will be rejected or preferred not according to their quality but according to the "quality" of the evaluation algorithm.

Since the number of bits (the costs) could be seen to be bounded by the input length itself, and all these numbers are integers, we can infer from Corollary 2:

Corollary 3. MDL-OPTIMAL-CODING *can be solved in time* $\mathcal{O}^*(2^n)$ *(or also* $\mathcal{O}^*(2^k)$*), where n denotes the number of input strings to be coded (and k upper bounds the coding costs).* □

Observe that $n < m$ in this particular application (even worse, m need not be bounded by any polynomial in n), since the input strings themselves are also always seen as possible regular expressions. So, the time bound derived by the preceding corollary is always superior to the trivial approach from Theorem 1.

Notice that similar coding methods are also used for data compression purposes. There, the "theory part" has sometimes to be explicitly encoded (as *side information*) and sometimes it is statically known to the (de)coder, which means that the "costs" of "opening up a facility" are zero. Hence, we face (again) the MEDIAN PROBLEM discussed in Section 3, so this problem is also parameterized tractable, when parameterized by an upper bound on the number of bits used for the compressed data.

4.2. Computational biology

Koivisto et al. described in [23] a method of applying ideas originating in the Minimum Description Length principle to identify *haplotype blocks* and to compare the strength of block boundaries. Intuitively, "a haplotype block can be considered to represent a sequence of ordered markers such that, for those markers, most of the haplotypes in the population cluster into a small number of classes. Each class consists of identical or almost identical haplotypes". (Quoted from [23].)

They propose a dynamic programming algorithm for the problem of computing an optimal block structure and estimating the probabilities of the block boundaries. This method relies on knowing a cost function whose computation is $\mathcal{NP}$-hard. However, this cost function (measuring the quality of the clusters/blocks) can be modeled with a p-MEANS PROBLEM and hence can be solved by the methods described in this paper. Since the weights involved grow at most exponentially with the input length, the fast subset convolution method can be used to further reduce the run times.

5. Conclusions and further research

We have started a systematic study of "facility location problems" and their variants, and their connections to other areas, from the parameterized and exact point of views. Many things are still to be done. We sketch some of these questions in the following.

(1) *Find better parameterized algorithms.* Our $\mathcal{FPT}$ algorithms are only a starting point. Are there better algorithms for these important problems? Can we do better for special cases, such as those related to p-means or p-medians? This quest for better algorithms and smaller kernels is also open for all the variants which we discuss in this paper; see also the following items.

(2) *Find more suitable parameters.* It is unclear whether the basic assumption in parameterized algorithmics, namely, that the parameter is only moderately large, is met in this set of problems. Are there different parameterizations that are more suitable in some applications? We give five examples of a different, possibly additional parameter:
   (2a) In [9], a modified scenario was considered where some "outliers" were permitted, i.e., customers that could not be served (at decent costs). This could deliver a natural small parameter.
   (2b) A kind of dual parameterization (compared to the previous sub-item) would be to assign weights to customers (modeling their importance) and to ask for serving a set of customers of weight at least r. Note that if all customers are given the same weight (say, one), then this question asks for a way to serve $\geq r$ customers, which means that we would allow for $n - r$ outliers in the terminology of item (2a).

(2c) In many situations, a company will already have opened a number of facilities, and the question is how to optimally improve the situation for the customers (and the company's budget) by opening up a small number of new facilities.

(3) *Exploit further properties of typical instances.* These may also lead to new useful parameterizations.

(3a) It is quite natural to assume that only (relatively) few facilities are within the "natural reach" of a single customer (it might be different from the viewpoint of the facilities, though). This implies a sort of degree restriction on the side of the customer (in the underlying bipartite graph) which might yield a better run time estimate of the dynamic programming algorithm; see [2].

(3b) In a number of areas we might hope for graph parameters such as treewidth to be small in applications; see [26] for the latest algorithmic developments in this area. Since Facility Location can be viewed as a graph problem where in practice many edges between customers and potential facility locations will be missing (or simply too expensive), such graph parameters might yield reasonable parameterizations.

(4) *Consider special cases and variants.*

(4a) In the capacitated facility location problem, each facility can serve at most c customers, giving rise to another natural parameter of the problem. Notice that capacitated problem variants are sometimes harder than their uncapacitated counterparts, at least from a parameterized perspective. For example, in [13], a capacitated variant of Vertex Cover was studied and shown to be harder than the uncapacitated (classical) variant, while [22] shows that, from the viewpoint of approximation, no differences in the quality of approximation algorithms can be observed for Vertex Cover.

(4b) Another aspect that has been neglected so far is geometry. Can we exploit metricity of costs to obtain better parameterized algorithms, as has been done in the case of approximation algorithms (see, e.g., [6])? The idea presented in the previous item is only one of several possible exploitations.

(5) *Study applications.* We already discussed two of them in the preceding section.

(5a) The Minimum Description Length principle (MDL) offers quite a general way of encoding information. So, given some abstract syntax for describing language-defining objects (like regular expressions that describe regular language in our example), we may look into the length of descriptions of elements of those languages (like words from regular languages) relative to language-defining objects (e.g., regular expressions), plus the size of encoding these objects themselves. Hence, MDL encodings (in general, as well as in the particular case discussed in this paper) seem to provide another quite special form of facility location problems. The special character of these problems would naturally depend on the chosen (description of the and via the) language-defining objects. Can we obtain better running times for parameterized/exact algorithms for these special cases?

(5b) There might be a broader connection to data compression; there, good p-means algorithms are crucial, e.g., in vector compression. The popular Lloyd algorithm (and variants thereof) may get stuck at local optima, so it might sometimes be a good idea to look for global optima. Notice that using exact algorithms might be a sensible approach here due to the slow convergence of Lloyd's algorithm in practical situations; see [15].

Acknowledgements

This paper is an extended version of the earlier published extended abstracts in [16,17]. We are grateful for the comments of the referees on these submissions.

The first author's research was supported by the Australian Research Council through the Australian Centre of Excellence in Bioinformatics.

Appendix. Explaining the coding with regular expressions

For a regular expression $r \in R$, $c(r)$ is some natural (direct) encoding of the word $c(r)$ over the alphabet Σ', which contains Σ and special letters like e (the empty word), $\cup$, ?, $*$, $+$. So, $|c(r)| = c_\Sigma|r|$ for some constant c_Σ, where $|r|$ simply denotes the length of the word r over Σ'.

As usual, $L(r)$ denotes the language described by the expression r; the standard recursive definition of $L(r)$ can be found in any textbook on formal languages.

For a string s and a regular expression r such that $s \in L(r)$, $c(s|r)$ is recursively defined as follows according to [21]:

- $c(s|s) = e$,
- $c(s|r_0 \cup \ldots \cup r_n) = b(i)c(s|r_i)$ if $s \in L(r_i)$,
- $c(e|r^*) = c(e|r?) = 0$,
- $c(s|r?) = 1c(r)$ if $s \neq e$,
- $c(s_1 \ldots s_k|r^*) = c(s_1 \ldots s_k|r^+) = 1^{\lceil \log_2(k+1) \rceil}0b(k)c(s_1|r) \ldots c(s_k|r)$ if $k > 0$ and $s_i \in L(r)$,
- $c(s_1 \ldots s_k|r_1 \ldots r_k) = c(s_1|r_1) \ldots c(s_k|r_k)$ for $k > 1$ if $s_i \in L(r_i)$.

Here, $b(i)$ yields the (usual) binary encoding of the integer number i. Observe that this recursive definition leaves some "degrees of freedom" according to a concrete implementation of c, since there may be several "decompositions" of

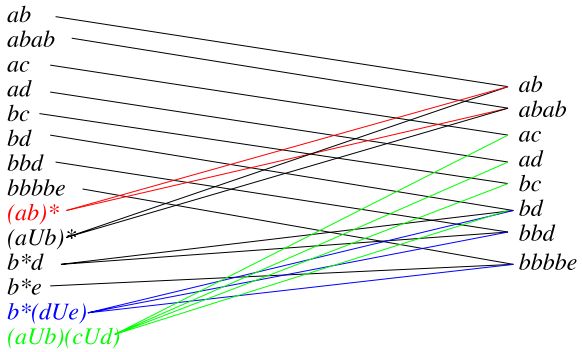

Fig. 1. How to apply the MDL principle.

(sub)strings and matching regular (sub)expressions. Our $\mathcal{NP}$-hardness proof of Theorem 8 does not depend on these technicalities. Moreover, observe that the reverse task of spelling out a word s given a code $c(s|r)$ is always uniquely possible, independently of the nondeterministic choices possibly made in the encoding process. Finally, a collection of regular expressions $R = \{r_1, \ldots, r_n\}$ can be viewed as disjunction of the member expressions, so $c(s|R) = c(s|r_1 \cup \cdots \cup r_n)$.

Example 1. As an example (taken from [20,21]), consider

$$\begin{aligned} c(abccabfggg|(ab \cup c)^*(de \cup fg^*)) &= c(abccab|(ab \cup c)^*)c(fggg|de \cup fg^*) \\ &= 1^3 0 b(4) c(ab|ab \cup c) c(c|ab \cup c)^2 c(ab|ab \cup c) 1 c(f|f) c(ggg|g^*) \\ &= 1^3 0100011011^2 0 b(3) \\ &= 11101000110111011. \end{aligned}$$

For the decoding procedure, given $r = (ab \cup c)^*(de \cup fg^*)$ as a theory, we know that the codeword $c = 11101000110111011$ can be decomposed as $c = c_1c_2$, where c_1 is encoded via $r_1 = (ab \cup c)^*$. According to the definition of the coding, c_1 is either 0 or starts with $1^{\lceil \log_2(k+1) \rceil} 0 b(k)$. This means that in our example, $\lceil \log_2(k+1) \rceil = 3$ and hence $k = (100)_2 = 4$. Hence, four codings with the help of the subexpression $ab \cup c$ follow the prefix 1110100 of c, so 0110 encodes *abccab*. This ends the processing of c_1, so it is now clear that $c_2 = 111011$. This part is encoded using the theory $de \cup fg^*$. The first bit of c_2 tells us that $c_2' = 11011$ is encoded using fg^*. Since f is fixed, this is the next letter to be output by the decoder. Finally, interpreting c_2' via g^* gives g^3, so the whole original sequence was reconstructed.

As an example for the decision problem, consider the following situation:

Example 2. Consider the strings

$$S = \{ab, abab, ac, ad, bc, bd, bbd\}.$$

As hypotheses "covering" some of these examples, we consider, besides the elements from S itself, the following regular expressions:

$$\{(ab)^*, (a \cup b)^*, b^*d, b^*e, b^*(d \cup e), (a \cup b)(c \cup d)\}.$$

Graphically, this covering relation can be depicted as a bipartite dominating set problem; see Fig. 1. Choosing, as indicated by the colorings, $R' = \{(ab)^*, b^*(d \cup e), (a \cup b)(c \cup d)\}$ as the theory, we get the following encoding lengths:

$$\begin{aligned} &c(ab|(ab)^*) = 101 \\ &c(abab|(ab)^*) = 11010 \\ &c(ac|(a \cup b)(c \cup d)) = 00 \quad \text{similar } ad, bc, bd \\ &c(bd|b^*(d \cup e)) = 1010 \\ &c(bbd|b^*(d \cup e)) = 110100 \\ &c(bbbbe|b^*(d \cup e)) = 11101001. \end{aligned}$$

Since $|c(bd|(a \cup b)(c \cup d))| = 2 < |c(bd|b^*(d \cup e))| = 4$, we choose the first interpretation. To indicate which "part" of the theory we select, we need two more bits per sample, so we need $3 + 5 + 4 * 2 + 6 + 8 + 2 * 8 = 46$ bits for coding of the whole sequence. Moreover, for the theory itself $\sum_{r \in R'} |r| \log_2(5 + 6) = (5 + 7 + 10) \log_2(11) = 22 * 4 = 88$ bits are needed.

By way of contrast, the "theory" $R'' = \{ab, abab, ac, ad, bc, bd, bbd, bbbbe\}$ alone would need $(2 + 4 + 2 + 2 + 2 + 2 + 3 + 5) \log_2(11) = 88$ bits to encode. To describe the samples with this theory, we would need $\log_2(8) = 3$ bits per

sample element, i.e., altogether 24 bits. So, currently, the naive theory which simply repeats the given items would be still advantageous (having an overall cost of $88+24 = 112$ bits). Adding the two further samples *ababab* and *bbbd* would change the picture: coding these samples with R' would need $5+6 = 11$ bits, so in total $88+46+11 = 145$ would be the incurred cost. The naive repetitive approach would generate $R''' = R'' \cup \{ababab, bbbd\}$ as the theory. Hence, the theory alone would need $(2+4+2+2+2+2+3+5+6+4)\log_2(11) = 128$ bits to be encoded. Describing each sample now takes $\log_2(10) = 4$ bits, so 40 more bits are necessary for computing all costs. Hence, now R' would become preferable.

Remark 2. From the point of view of data transmission, it is not quite justifiable why transmitting a theory R consisting of say ℓ regular expressions $r_1, \ldots, r_\ell$ "only" costs $c(R) = \sum c(r_i)$ bits, since delimiters between the singular regular expressions would also always have to be transmitted. So, it would be "fairer" to set

$$c'(R) = c(r_1 \cup r_2 \cup \cdots \cup r_\ell) = c(R) + (\ell - 1)\log_2(|\Sigma| + |\mathcal{M}|).$$

This will make a difference, in particular for small examples. Coming back to Example 2, the theory R' would actually take $c'(R') = (5+7+10+2)\log_2(11) = 96$ bits, leading to an overall coding cost of $96+46 = 142$ bits. The naive theory R'' would then cost $c'(R'') = c(R'') + 7\log_2(11) = 88+28 = 116$ bits, yielding an overall cost of $116+24 = 140$ bits. This is still slightly better than for R', but the difference is now really marginal.

To finalize that example, let us consider a "compromise theory":

$$\bar{R} = \{(ab)^*, bbd, bbbe, (a \cup b)(c \cup d)\}.$$

As with R', explaining what part of the theory to select takes two bits. Describing *bbd* and *bbbe* with its "best-fitting" expressions now takes no bits at all, so 32 bits suffice to encode the whole sequence. The theory itself would cost $c(\bar{R}) = (5+7+3+4)\log_2(11) = 76$ bits, amounting to an overall cost of $32+76 = 108$ bits, which is already slightly better than the naive theory advocated before. Measured in terms of c', $c'(\bar{R}) = 76+3\times 4 = 88$, which leads to an overall cost of $88+32 = 120$ bits, clearly better than those for the two alternatives R and R'.

The reader should finally observe that the suggested change from c to c' (in contrast to the case for [21]) will not dramatically shatter the $\mathcal{NP}$-completeness construction shown in the proof of Theorem 8.

References

[1] A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, Fourier meets Möbius: fast subset convolution, in: STOC, ACM Press, 2007, pp. 67–74.
[2] A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, Trimmed Möbius inversion and graphs of bounded degree, in: S. Albers, P. Weil (Eds.), Symposium on Theoretical Aspects of Computer Science STACS, in: Internationales Begegnungs- und Forschungszentrum für Informatik, IBFI, Schloss Dagstuhl, Germany, 2008, pp. 85–96.
[3] H.L. Bodlaender, R.G. Downey, M.R. Fellows, D. Hermelin, On problems without polynomial kernels, Journal of Computer and System Sciences 75 (2009) 423–434.
[4] H.L. Bodlaender, S. Thomassé, A. Yeo, A. Fiat, P. Sanders, Kernel bounds for disjoint cycles and disjoint paths, in: Algorithms—ESA 2009, 17th Annual European Symposium, in: LNCS, vol. 5757, Springer, 2009, pp. 635–646.
[5] A. Bumb, Approximation algorithms for facility location problems, Ph.D. Thesis, Univ. Twente, The Netherlands, 2002.
[6] J. Byrka, K. Aardal, An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem, SIAM Journal on Computing 39 (6) (2010) 2212–2231.
[7] M. Cesati, The Turing way to parameterized complexity, Journal of Computer and System Sciences 67 (2003) 654–685.
[8] M. Charikar, S. Guha, E. Tardos, D.S. Shmoys, A constant-factor approximation algorithm for the *k*-median problem, Journal of Computer and System Sciences 65 (2002) 129–149.
[9] M. Charikar, S. Khuller, D.M. Mount, G. Narasimhan, Algorithms for facility location problems with outliers, in: ACM-SIAM Symposium on Discrete Algorithms SODA, 2001, pp. 642–651.
[10] D. Conklin, I.H. Witten, Complexity-based induction, Machine Learning 16 (1994) 203–225.
[11] L.E. Dickson, Finiteness of the odd perfect and primitive abundant numbers with *r* distinct prime factors, American Journal of Mathematics 35 (1913) 413–422.
[12] M. Dom, D. Lokshtanov, S. Saurabh, Incompressibility through colors and IDs, in: S. Albers, A. Marchetti-Spaccamela, Y. Matias, S.E. Nikoletseas, W. Thomas (Eds.), Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Part I, in: LNCS, vol. 5555, Springer, 2009, pp. 378–389.
[13] M. Dom, D. Lokshtanov, S. Saurabh, Y. Villanger, Capacitated domination and covering: a parameterized perspective, in: M. Grohe, R. Niedermeier (Eds.), Parameterized and Exact Computation, Third International Workshop, IWPEC, in: LNCS, vol. 5018, Springer, 2008, pp. 78–90.
[14] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, 1999.
[15] Q. Du, M. Emelianenko, L. Ju, Convergence of the Lloyd algorithm for computing centroidal Voronoi tessellations, SIAM Journal on Numerical Analysis 44 (2006) 102–119.
[16] M.R. Fellows, H. Fernau, R. Fleischer, J. Xu, Facility location problems: a parameterized view, in: Algorithmic Aspects in Information and Management AAIM 2008, in: LNCS, vol. 5034, Springer, 2008, pp. 188–199.
[17] H. Fernau, Extracting minimum length document type definitions is NP-hard, in: G. Paliouras, Y. Sakakibara (Eds.), Grammatical Inference: Algorithms and Applications; 7th International Colloquium ICGI, in: LNCS/LNAI, vol. 3264, Springer, 2004, pp. 277–278.
[18] L. Fortnow, R. Santhanam, Infeasibility of instance compression and succinct PCPs for NP, in: C. Dwork (Ed.), ACM Symposium on Theory of Computing, STOC, ACM, 2008, pp. 133–142.
[19] M.R. Garey, D.S. Johnson, L. Stockmeyer, Some simplified NP-complete graph problems, Theoretical Computer Science 1 (1976) 237–267.
[20] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, XTRACT: a system for extracting document type descriptors from XML documents, in: Proceedings of ACM SIGMOD Conference on Management of Data, 2000, pp. 165–176.
[21] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, XTRACT: learning document type descriptors from XML document collections, Data Mining and Knowledge Discovery 7 (2003) 23–56.
[22] S. Guha, R. Hassin, S. Khuller, E. Or, Capacitated vertex covering, Journal of Algorithms 48 (2003) 257–270.
[23] M. Koivisto, H. Manila, M. Perola, T. Varilo, W. Hennah, J. Ekelund, M. Lukk, L. Peltonen, E. Ukkonen, An MDL method for finding haplotype blocks and for estimating the strength of block boundaries, in: R.B. Altman, A.K. Dukner, L. Hunter, T.A. Jung, T.E. Klein (Eds.), Pacific Symposium on Biocomputing 2003, PSB 2003, 2002, pp. 502–513.

[24] B. Mishra, Algorithmic Algebra, in: Texts and Monographs in Computer Science Series, vol. XIV, Springer, 1993.
[25] D.B. Shmoys, É Tardos, K. Aardal, Approximation algorithms for facility location problems (extended abstract), in: 29th ACM Symposium on Theory of Computing, 1997, pp. 265–274.
[26] J.M.M. van Rooij, H.L. Bodlaender, P. Rossmanith, Dynamic programming on tree decompositions using generalised fast subset convolution, in: A. Fiat, P. Sanders (Eds.), Algorithms—ESA 2009, 17th Annual European Symposium, in: LNCS, vol. 5757, Springer, 2009, pp. 566–577.
[27] N.E. Young, *k*-medians, facility location, and the Chernoff–Wald bound, in: ACM-SIAM Symposium on Discrete Algorithms SODA, 2000, pp. 86–95.